View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Waseda University Repository

2005 年度 修士論文

初心者プログラミング学習補助法
～段階的プログラミング学習～

提出日：2005 年 2 月 2 日

指導：筧捷彦教授

早稲田大学 理工学研究科情報ネットワーク専攻
学籍番号：3603u1128

日野 孝昭

目 次

第1章　概要

初心者がプログラミングを学習する場合, 最初は簡単な概念を学び, 次第に高度な概念を習得するという順序を踏むのが一般的である. 最初は, 難しい概念を知らなくても大丈夫な言語や環境で学習し, 学習が進むに連れて高度な概念を必要とする Java のような実用的な言語へ移行させるのがよい.

これまで, われわれはプログラミング初心者が Java を学習する手助けになるプログラミング環境 NigariSystem を開発した. NigariSystem は Java 言語を学習する一歩前の段階に用いる環境として開発されている. Java 言語習得を目的とする授業の導入に NigariSystem を用いる、試用授業をおこなった [2], [6], [7]. 試用授業では, オブジェクトの可視化機能や, 簡素化された言語についてもある程度の評価が得られた. しかし, 簡単すぎる言語仕様のため, 導入が終わり Java 言語に触れてみると Nigari 言語とのギャップに戸惑っている学生が多く見られた. また, オブジェクトの可視化機能で描画することができる情報が, プログラムの振る舞いを理解するためには不十分であり, デバッグに手間取る学生も見られた.

本研究では, Java 言語と Nigari 言語のギャップを緩和する機能と, NigariSystem の可視化機能を強化した機能を備えた NJava を開発した. 早稲田大学コンピュータ・ネットワーク学科一年の前期のプログラミングの授業に適用し効果を調べた. この授業で NJava を用いて学生に Java 言語を習得してもらった. 実験では, Java 言語と Nigari 言語のギャップを緩和することができた. また, 強化した可視化機能についてもデバッグに役立ったという反応も得た.

第2章　プログラミング初心者が学習上陥る問題

2.1　プログラミング環境が原因で陥る問題

プログラミングをする際, ある環境ではコンパイルをし, 実行することができるが, 他の環境になるとできない場合がある. 初心者がこの状況に直面した場合, なぜコンパイルをしたり, 実行したりすることができないかを理解し, 修復することは難しい.

また, プログラミング初心者はパソコンを操作するのも初心者である場合がある. この場合, 自分が作成したプログラムがどこに保存されているかすらわからない場合が多い. 以前に作ったプログラムの行方がわからずまた一からやり直しとなるとやる気を失うことは当然である.

2.2　プログラミング言語仕様から陥る問題

現在の初心者プログラミング教育では, 初めて触れるプログラミング言語に複雑な言語仕様のプログラミング言語を用いていることが多い. そして, 初心者が理解できないで部分は, おまじないとして済ませている. これでは, 自分が書いたプログラムであるにもかかわらず、理解できない部分が存在してしまう. 初心者は, こういった状況を強制的にプログラミング学習をやらされていると感じてしまい学習意欲を失ってしまう.

また, 簡単な言語仕様のプログラミング言語で基本的なプログラムの仕組みを学習し, その後複雑な言語仕様のプログラミング言語に触れるという学習方法もある. この場合は簡単な言語仕様のプログラミング言語で基本的なプログラムの仕組みを学習するまでは順調である. しかし, その後の複雑な言語仕様のプログラミング言語に触れると, いままでとのギャップに驚いてしまう.

2.3　プログラム実行時の状況がわからないために陥る問題

プログラミング初心者は記述したプログラムがどういう流れで実行が進んでいくかという基本的な知識を持っていない. それにもかかわらず, 初心者が触れるサンプルプログラムの多くは, プログラムを実行し結果のみを表示するものが多いので, 初心者がプログラムの処理される順序を理解することが困難である. これではプログラミング初心者が基本的な知識を学習するには不親切すぎる. また, プログラムが処理されているのを確認するために, プログラムの所々で出力を用いて情報を表示することもある. Java でいうならば System.out.println を用いて一時的, 部分的情報を表示するといったことである. これで得られる出力は, プログラムの一部が処理されどのように影響を及ぼしたかを知るには不十分な情報でしかない.

第3章　研究目的と概要

3.1　プログラミング環境が原因で陥る問題の解消

プログラミング教育には, 教育を受ける側が自分の手で作業をする必要性がある. そのため, 計算機の環境は非常に重要な要素といえる. 初心者が, ある環境では動作するが, 他の環境では動作しないという状況に陥った場合, 自分で原因を究明し, 自分で修復することは困難である. この問題を回避し, 環境に依存しないプログラミング学習システムにするため, Java 言語で実装する.

初心者はファイルの管理を失敗しファイルをどこに置いたかがわからなくなる場合がある. 以前作ったファイルを利用したい場合にも, 無駄な労力を費やして作成しなければならない. 初心者の学習の上では無駄な労力を費やし学習意欲を失うといったことは極力避けなければならない問題である. こういった失敗をなくすため, 教材で保存する場所を指定したり, システムで強制的に保存場所を指定したりする.

3.2　プログラミング言語仕様が原因で陥る問題の解消

プログラミング初心者は, いきなり複雑な記述方法に触れると学習意欲を失う. 初心者が一番初めに触れる言語としては, 不必要な機能, 複雑な文法や特徴を極力除き, 覚えるという作業を必要としないものがよい. 複雑な記述方法を排除し, 理解できるプログラムのみを書けばよいようにする. そういったことを踏まえて, 覚えることの少ない仕様の言語を作成する.

また, 簡単な仕様のプログラミング言語で学習しただけでは, プログラムを習得したとはいえない. 複雑な仕様のプログラミング言語を学習する必要がある. しかし, すぐに複雑な仕様のプログラミング言語に移るには問題がある. それは, 複雑な言語仕様のプログラミング言語には複雑な記述方法や概念は多々あるためである. そのひとつがわかれば, 実際に動くプログラムが書けるというものではない. そのため, 実際に動くプログラムを書くには複数の複雑な記述方法や概念を理解していなければならない. プログラミング初心者にそういった理解が難しいものを複数一度に学習してもらうことは困難である. そこで, おまじないを必要としない言語仕様のプログラミング言語を学習してから, 段階をおって複雑な言語仕様のプログラミング言語を学習する方法について考える. この方法を実践する場合, 段階をどのように設定するかが重要である. Nigari を用いた実験結果などから効果的な段階の踏み方を考える.

3.3　プログラムの処理の進み方の理解の手助け

プログラム初心者にとって, プログラムを実行して結果がコンソールに出力されるというだけでは味気ない. もしこれが続くならば, 学習意欲が低下してしまうだろう. そこで, プログラムの動きが目で見えるようにする方法を考える. 実行中のオブジェクトの状態内容によって画面上をオブジェクトが動き回るなど, 方法を考える. そして, 簡単なプログラムから複雑なプログラムまで, 実

行したときの状態の変化を目で見てわかりやすいようにする.

　初心者のプログラミング学習においてプログラムがどのような順序で処理されているかを理解することは重要なことである. 実行時, 処理されているプログラムの場所がわかる仕組みがあれば, 初心者でもプログラムの処理されている順序を理解しやすいはずである. また, 初心者がよく陥る状況としてコンパイル, 実行はできるが自分が期待したものとは違う結果が返されることがある. こういった場合, プログラムのどの場所が処理されていて, オブジェクトの状態がどのようになっているかを知ることができればデバックに役立つはずである.

第4章　これまでの研究

4.1　NigariSystem とは

昨年まで私は, 3. であげた研究目的を踏まえて NigariSystem を作成し, 初心者プログラミング教育補助における効果の研究を行っていた. NigariSystem の見た目は図 4.1 の通りである. NigariSystem は, メインウィンドウ, オブジェクトインスペクタ, エディタ, コンソールからなっている.

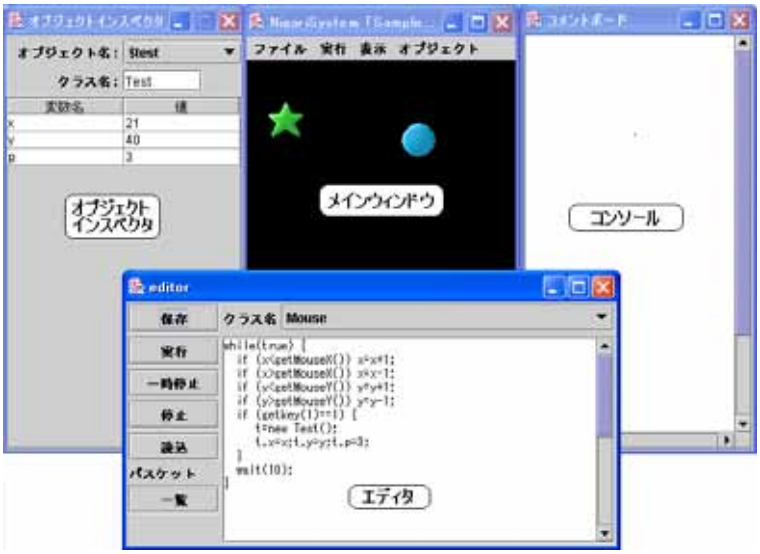

図 4.1: 題材ごとの難易度

- メインウィンドウ
  オブジェクトを表示するためのウィンドウ. 実行時に, このメインウィンドウ上を設計されたプログラムに応じてオブジェクトが動き回る.
- オブジェクトインスペクタ
  オブジェクトが持つ変数の値を見ることができるウィンドウ.
- エディタ
  オブジェクトの動作記述であるプログラムを編集するために使うウィンドウ.
- コンソール
  プログラムを実行したときに出力が表示されるウィンドウ.

4.1.1 初心者に優しい環境

NigariSystem は, Java で作成した. こうすることにより, 初心者は JavaVm が搭載されている PC であれば, 簡単に NigariSystem を Web 上からダウンロードし利用することができる。

プログラムを書いたファイルの保存場所を NigariSystem で管理するようにした. ユーザーはファイルをどこに保存したかをわざわざ探すことなく, 簡単に見つけ出すことができる.

4.1.2 簡単な言語仕様

初心者は, 理解に困ることのない簡単な言語仕様である Nigari 言語を用いてプログラムを書くことができる. Nigari 言語では, Java 言語や C 言語に用いられているメジャーな構文をピックアップした. これにより、初心者は, 簡単な Nigari 言語を用いることでメジャーな構文を学習することができる。

4.1.3 簡単にアニメーションを作れる機能

プログラムを実行すると, オブジェクトは自分自身が持っている変数 x と y の値によりメインウィンドウを移動する. これにより, プログラムの動きを目で観察することができる.

4.2 NigariSystem を用いた実験

初心者向けのプログラミング環境 NigariSystem の効果を調べるため, 確認の授業で用いてみた. 2003 年度早稲田大学理工学部コンピュータ・ネットワーク工学科の一年生を対象とした。まず, 4/12 から 6/9 までの 8 回の授業で NigariSystem を用いてプログラミングを学習してもらう。そして, 基本的な構文や概念を学習してもらった後, 6/16 から 7/7 までの 4 回の授業で Java を用いてプログラミングの学習をしてもらう。

- **対象者**
  - 2003 年度早稲田大学理工学部コンピュータ・ネットワーク工学科一年
  - 人数 240 名
  - プログラム初心者, 経験者混合
- **授業内容**
  - 4/12 講義概要
  - 4/21 PC の使い方とプログラミング
  - 4/28 Nigari のインストールと試用
  - 5/12 変数/while 文
  - 5/19 while 文 (つづき)/if 文
  - 5/26 if 文 (つづき)/マウス入力
  - 6/02 複数のオブジェクト/マルチスレッド

– 6/09 メソッド/オブジェクトの実行時生成
– 6/16 Java の概要/変数の宣言/while 文 (この日から Java)
– 6/23 制御構造/メソッド/文字列
– 6/30 配列/コマンドライン引数/並べ換え
– 7/07 並べ換え (つづき)/文字入力
– 7/14 補講 (クラス試験)

4.3 NigariSystem を用いた授業の結果

4.3.1 初心者に優しい環境の成果

Java でシステム全体を作成し, Java さえインストールされていれば簡単に利用できるようにした. Web ページからダウンロードするだけで簡単に利用することができる. ダウンロードした場所がわからないといった問題が生じたものの, 一度使ったあとでは, 環境によるプログラミング学習への妨げは生じなかった.

しかし, プログラムのバージョン管理のときに問題が生じた. 例えば, はじめに Rectangle クラスのプログラムを Rectangle.java ファイルに書いてあるとする. その後, 学生が Rectangle クラスのプログラムを変更し, そのクラス名を Rectangle1 として同じ Rectangle.java ファイルに保存する. コンパイル後, Rectangle クラスを実行する. 学生は Rectangle クラスの実行結果として、Rectangle1 クラスのプログラムの内容がを期待している. しかし, Rectangle クラスを実行しているので変更が加えていない結果が返ってくる. そして, 学生はプログラムの変更が実行に反映されていない原因がわからないという状況があった.

4.3.2 簡単な仕様の言語を用いた学習の考察

2003 年度の授業では, Nigari 言語を用いてプログラムの基本的知識を学習してもらい, その後に複雑な言語仕様の Java 言語を学習してもらった. この方法について賛否を学生アンケートで調べたところ, 簡単な仕様の言語で基本的な知識を学習し, 続いて複雑な仕様の言語を学習するという方法には多くの学生から賛同を得られた. しかし, 二つの問題点が感じられた.

一つ目の問題点は, Nigari 言語から Java 言語に移ると、次のような Java 言語の多くの決まりごとに, 学生はギャップを感じてしまう。

- 変数を宣言しなければならない
- プログラム全体を “class XXXXX{” と “}” で囲わなければならない
- メインルーチンを “public static void main(String args[]){” と “}” で囲わなければならない。

Java に移ってから学習意欲が低下していく様 [6] を見る限り, いきなり多くの決まりごとに直面するのは初心者のプログラミング教育には適していない. そこで, 複数の決まりごとに一度に直面するのではなく, 少しずつ段階を追って接していくようにして, このギャップを緩和するべきである.

二つ目の問題点は, Nigari 言語の独特の記述方法を学んでいるため, Java 言語に移ったときにその記述方法でプログラムを書いてしまう点である. Nigari 言語は Java 言語に似せて作られているとはいえ, 細部には異なっている部分がある. 例をあげると, コンストラクタやメソッドの記述方法

である. Java 言語に移行してからも Nigari 言語独特の記述方法でプログラムを書く人たちがいた. Java 言語でも Nigari 言語と同じように記述できる部分とできない部分があることが原因で起きている問題である. これを解決するには, Nigari 言語の独特の記述方法を必要とする部分は, Java 言語に移ってから学習するようにすればよいと考える. そのためにも, 学習する題材の順番を練り直す必要がある.

4.3.3 NigariSystem での可視化機能の考察

NigariSystem での可視化機能は, 変数 x と変数 y に入っている値によって, 画面をオブジェクトが移動するというものだった. 変数として x と y のみを用いるだけでいいような簡単なプログラムのうちは, プログラムの動きを確認するのにいい影響がみられた. しかし, 複雑なプログラムになってきた場合は, 知りたい情報は変数 x と y の値だけではない. そのため, 目で見られる情報が x と y だけである Nigari の可視化機能では不十分である. また, コンパイラが通り, 実行してみたプログラムが, 自分が期待したものとは違う結果を返したとき, これだけの情報では間違いを探すことができない. オブジェクトの状態情報を表示する方法を追加したほうがいいように思える. そうした場合, 表示が複雑になることが考えられるので, 現在実行中のプログラム中の位置がわかるようになればいいと思える. 以後, 現在実行されているプログラム中の位置を, プログラムの実行位置と呼ぶ.

4.4 NigariSystem を用いた実験から重視すべきと思われたこと

4.4.1 言語仕様の違いによる混乱の削除

2003 年の授業で用いた方法は, 簡単な仕様である Nigari 言語で基本的な文法を学んでから, いきなり Java 言語を学習してもらうという方法だった. 簡単な仕様の Nigari 言語と複雑な仕様の Java 言語の二段階を追った学習方法である. 簡単な言語から複雑な言語へと進むというこの方法は, 初心者のプログラミング学習としては効果があるように思える. この方法では, スムーズに Java 言語に移行できた人はごく一部で, 多くの人が Nigari 言語と Java 言語のギャップに戸惑っていた. そこで, Nigari 言語で学習したあとに, すぐ Java 言語を用いて学習するのではなく, Nigari 言語より Java 言語に似ている言語を間にはさんで学習することして, このギャップを緩めることが重要だと考える.

Nigari 言語は Java 言語をまねて作られている. しかし, 細部は異なっている. 特に異なっているのは, コンストラクタやメソッドの記述方法である. この違いのために, Java 言語に移ったときも Nigari 言語の記述方法でコンストラクタやメソッドを記述する人たちが目立った. こういった混乱を避けるためにも, Nigari 言語を用いた学習では, Nigari 言語独特の記述を必要としない題材のみを学習すべきである.

4.4.2 プログラムの可視化機能の強化

NigariSystem の可視化機能は, オブジェクトの変数 x と変数 y に入っている値によって, 画面をオブジェクトが移動するというものである. 変数が x と y といった二つで問題ない簡単なプログラムのうちは, 今の NigariSystem の可視化機能で十分である. しかし, プログラムが複雑になってき

た場合, 今の NigariSystem の可視化機能から得られる情報では学生が, デバックをしたり, プログラムの動きを目で追ったりするには不十分である. デバックをする場合に必要な情報は, 実行位置とその時のオブジェクトの状態情報すべてである. そこで, 実行位置とそのときのオブジェクトの状態情報すべてを目で見ることができる可視化機能にする必要があると考える.

第5章　４つのレベルに分けたプログラミング学習

5.1　NigariSystem を使った実験からの考察

5.1.1　学生が書いたプログラムから考える段階分け

Nigari 言語を用いた学習から Java 言語を用いた学習へと移行する方法は, Nigari 言語独特の記述方法に注意を払う必要がある. 学生が混乱する原因となっていた Nigari 言語と Java 言語の記述方法の違いには, “コンストラクタの記述方法”, “メソッドの記述方法”, “配列の作り方” などがある.

メソッドの記述方法を例にあげる. メソッドの記述方法の違いは図 5.1 が示すようである. Java 言語の場合はメソッドの名前の前に, “public void” のように, 修飾子や返り値の型を記述する. しかし, Nigari 言語の場合は, メソッドの名前の前には “function” と書くだけでよい. この違いになじめず, 学生は Java 言語でプログラムを書いている場合でも, メソッドの定義を記述する時に, “public void” と書かずに, “function” と書いている学生がいた。

このように, Java 言語と記述方法に違いがあるものを Nigari 言語を用いている段階で学習させると, 余計な混乱が生じる. Nigari 言語を用いる段階では, Java 言語の記述方法と違う記述方法が必要な題材は学習させないほうがよい.

そうすると, Nigari 言語を用いての学習は while 文, if 文までにするべきである.

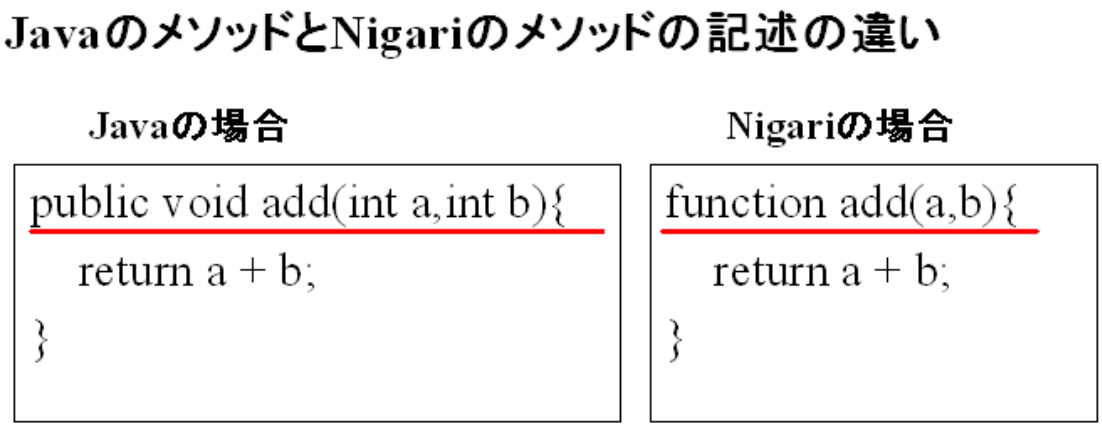

図 5.1: java のメソッドと Nigari のメソッド記述方法の違い

5.1.2 アンケート結果から考える段階分け

対象とする授業で, 学生に学習してもらう題材は, 反復 (while 文), 分岐 (if 文), 変数宣言, 変数の型, 配列, オブジェクト, メソッド, ローカル変数, デバッグである. これらの題材を, 学生が NigariSystem を用いて学習しているときにどのように感じていたかをまとめたものが図 5.2 と図 5.3 である. 難しいと思った題材は, 図 5.2 を見てわかるとおり変数, if 文, メソッド, 配列であった. これらの題材ごとに段階を準備するのはさすがに多すぎる。そこでこれらの中から, 段階を踏むべき題材を絞り込む必要がある. 題材ごとに出した問題を自分の力で解くことができたかを尋ねたアンケートの集計を図 5.3 に示す。これを見てわかるとおり, 配列, メソッドがずば抜けて解けていなかった. このことから, Nigari 言語から Java 言語に移る際に踏むべき段階は配列とメソッドの二つに絞ればいいように考えられる.

そこで, まず第 1 段階は Nigari 言語で基本的な文法について学習する. ここで学習する題材は, 反復 (while 文), 分岐 (if 文) とする. このときに書くプログラムの段階を以後 1 レベルと呼ぶ. プログラムを実行するときに使う VM は, Nigari 言語の VM を使用する.

つづいて, 第 2 段階は 1 レベルのプログラムに変数宣言を追加する. ここで, 変数宣言を学習するとともに, 変数の型や配列も学習する. このときに書くプログラムの段階を以後, 2 レベルと呼ぶ. プログラムを実行するときに使う VM は, Java 言語の VM を使用する. 2 レベルから Java 言語の VM を使う理由は、Nigari 言語には型という概念が存在せず, 変数宣言をしなくてもコンパイル, 実行ができてしまい, 型の学習をするのに Nigari 言語は適していないためである。この段階で学生が書いたプログラムは, Java 言語のプログラムとして不十分でありコンパイルに失敗する。そのため, Java 言語の言語仕様にあったプログラムになるように足りない部分をシステムで作るといった工夫が必要である.

第 3 段階は, 2 レベルで書くプログラムに, メソッドの概念を追加する. ここで, オブジェクト, メソッド, ローカル変数という題材を学習する. このときに書くプログラムの段階を以後レベル 3 と呼ぶ. プログラムを実行するときに使う VM は, Java 言語の VM を使用する. レベル 3 で書かれたプログラムも, レベル 2 同様 Java のプログラムとして不十分である。そのため, Java 言語の言語仕様にあったプログラムになるようにする工夫が必要である.

第 4 段階は, Java 言語でプログラムを書く. ここでは, デバックについて学習する. このときに書くプログラムの段階を以後, 4 レベルと呼ぶ. プログラムを実行するときに使う VM は, Java 言語の VM を使用する.

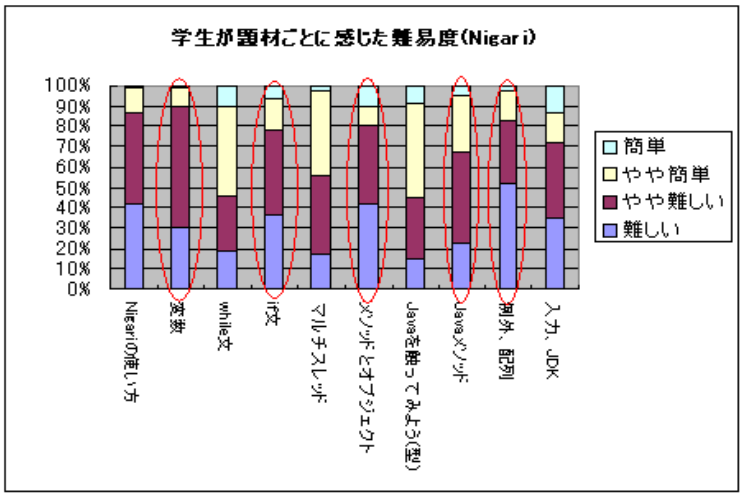

図 5.2: 題材ごとの難易度

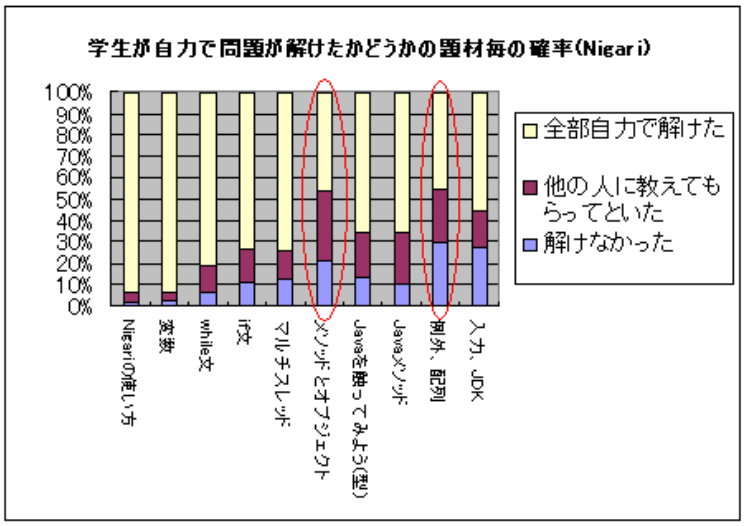

図 5.3: 題材ごとの問題が自分で解けたか否かの確率

5.2　各レベルで記述するプログラム

5.2.1　1 レベルで記述するプログラム

1 レベルで書くプログラムは図 5.4 のように, Nigari の言語仕様にそったプログラムである. Nigari 言語にもメソッドやコンストラクタを記述するための構文が用意されているが, 1 レベルでプログラムを学習している間には, メソッドやコンストラクタは学習しない.

```
x = 0;
y = 0;
while(x < 100){
    x++;
    y++;
}
```

図 5.4: 1 レベルで記述するプログラム

5.2.2　2 レベルで記述するプログラム

2 レベルで書くプログラムは図 5.5 のように, 1 レベルで書くプログラムに変数宣言をする. 変数の宣言は変数宣言部に記述し, 1 レベルで書いていたプログラムをメインルーチン部に記述する。

```
変数宣言部
int x,y;
メインルーチン部
x = 0;
y = 0;
while(x < 100){
    x++;
    y++;
}
```

図 5.5: 2 レベルで記述するプログラム

5.2.3　3 レベルで記述するプログラム

3 レベルで書くプログラムは図 5.6 のようになっている。これを, を “class XXXXX{” と “}” で囲うと Java 言語のプログラムになる.

```
int x,y;
public void run(){
   x = 0;
   y = 0;
   while(x < 100){
      x = add(x);
      y = add(y);
   }
}
public int add(int i){
   return i + 1;
}
```

図 5.6: 3 レベルで記述するプログラム

5.2.4　4 レベルで記述するプログラム

4 レベルで書くプログラムは図 5.7 のように Java 言語仕様に沿ったプログラムである.

```
public class Program extends Thread{
    int x,y;
    public void run(){
        x = 0;
        y = 0;
        while(x < 100){
            x = add(x);
            y = add(y);
        }
    }
    public int add(int i){
        return i + 1;
    }
}
```

図 5.7: 4 レベルで記述するプログラム

第6章　段階を追ったプログラミング学習システム“NJava”の実装

NJava とは NigariSystem を用いた実験結果を踏まえ, NigariSystem を改良した初心者プログラミング教育環境ソフトである.

6.1　プログラム解析機構

NJava では, 一つのクラスにつき, Nigari で実行するためのファイル (拡張子 “.nigari”) と Java で実行するためのファイル (拡張子 “.java”) の二つを準備する. 例えば, Test クラスを新規作成したときは, 図 6.1 のような二つのファイルを作る. Nigari ファイルの中には何もかかれていない. Java ファイルは, Thread クラスを継承し, run メソッドのみが記述されているプログラムが書かれているファイルが作られる. Java ファイルが Thread クラスを継承しているのは, 学生が簡単にマルチスレッドプログラムを作ることができるようにするためである. 通常 Java の Test クラスを実行する場合は Test クラスの main メソッドが実行されるが, NJava の場合は Test クラスの run メソッドが実行される.

二つのファイルは, 一方が変更された場合は, 他方にもその変更を反映させる. この作業を助ける機を持ったプログラム解析機構をシステムに準備した. プログラム解析機構では, Java のプログラムを図 6.2 のようになっているプログラムを取り扱う. 図の各部分は, 次の表に示すとおりの情報が書かれているものとする。

- **A 部分**　import 情報
- **B 部分**　インスタンス変数宣言
- **C 部分**　メインルーチン
- **D 部分**　クラスの中身

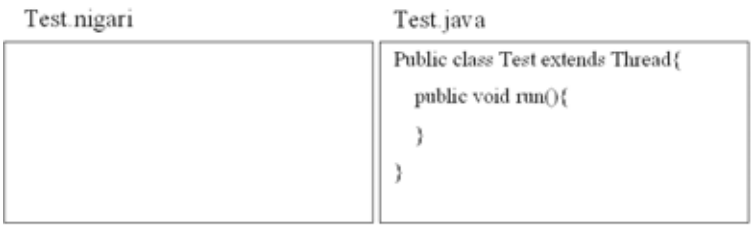

図 6.1: クラスの新規作成時

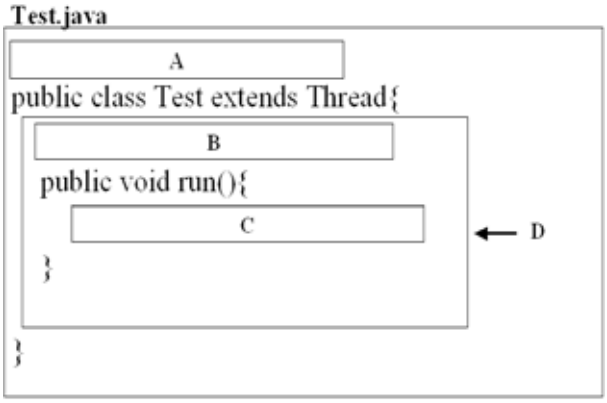

図 6.2: Progarm クラスの Java ファイル解析

6.2 プログラムの保存方法の実装

これ以降, n レベルのプログラムを編集するエディタのことをレベル n エディタという.

6.2.1 レベル 1 エディタでプログラムを保存する場合

レベル 1 エディタで保存する方法を図 6.3 に示す. 1 レベルで書かれているプログラムは Nigari 言語の仕様に従って書かれてたものである.

Nigari ファイルには, レベル 1 エディタに書かれている内容をそのまま保存する.

Java ファイルには, 保存する前のファイルをプログラム解析機構を用いて解析し, C 部分をエディタに書かれている内容に変更して保存する. Java ファイルにも変更を反映させるのは, レベル 2 以上のエディタで同じクラスのプログラムを編集することになった時に, 同じプログラムをもう一度書かなくてすむようにするためである.

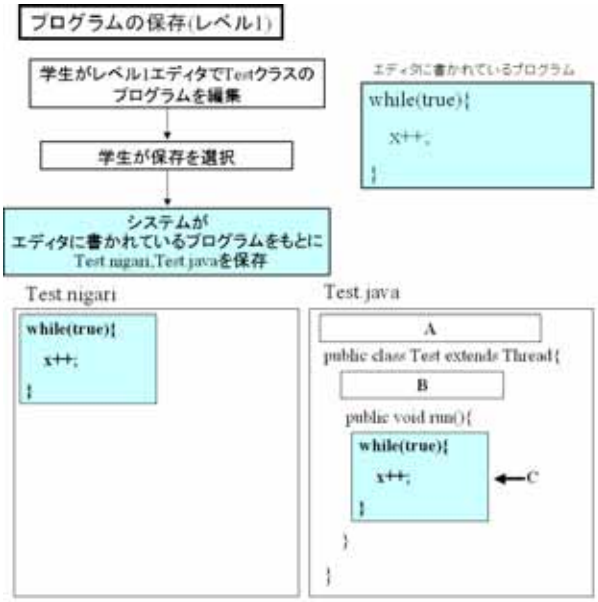

図 6.3: レベル 1 エディタでプログラムを保存する場合

6.2.2 レベル 2 エディタでプログラムを保存する場合

レベル 2 エディタで保存する方法を図 6.4 に示す. レベル 2 エディタは変数宣言部とメインルーチン部に分かれている.

Nigari ファイルには, メインルーチン部に書かれている内容をそのまま保存する.

Java ファイルには, 保存する前のファイルをプログラム解析機構を用いて解析し, B 部分を変数宣言部に書かれている内容, C 部分をメインルーチン部に書かれている内容に変更して保存する.

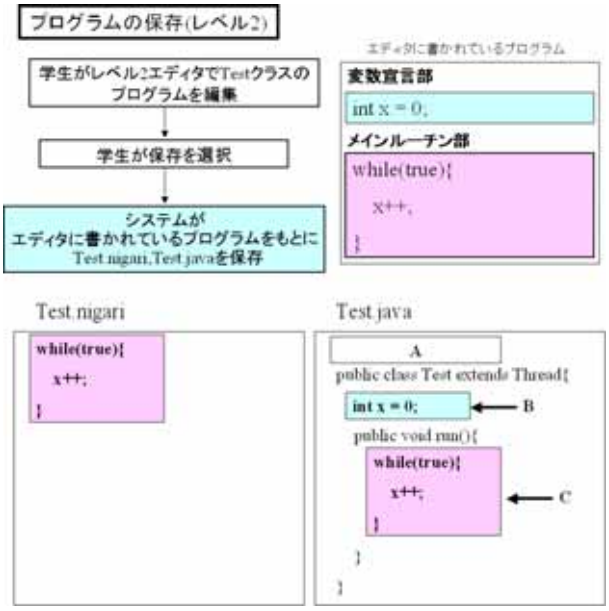

図 6.4: レベル 2 エディタでプログラムを保存する場合

6.2.3 レベル 3 エディタでプログラムを保存する場合

レベル 3 エディタで保存する方法を図 6.5 に示す.

Java ファイルは, 保存する前のファイルをプログラム解析機構を用いて解析し, D 部分をエディタに書かれている内容に変更して保存する.

Nigari ファイルは, 保存された Java ファイルをプログラム解析機構を用いて解析し, C 部分に相当する内容に変更して保存する.

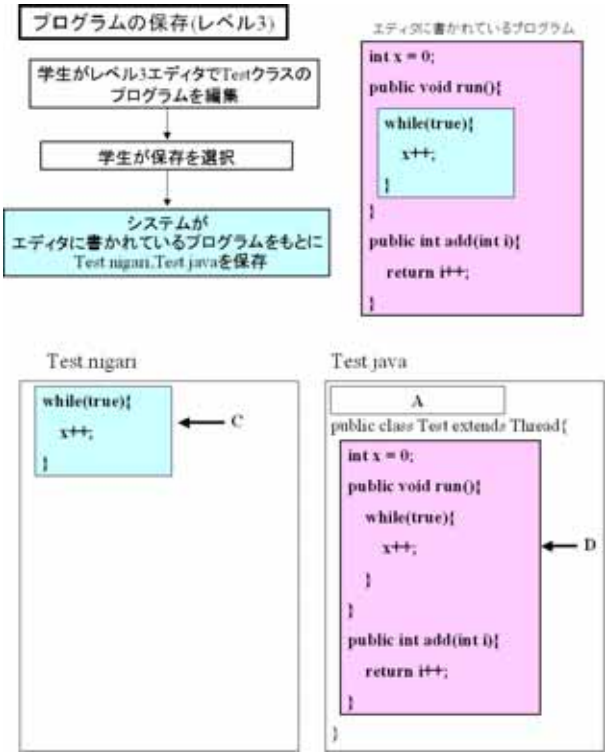

図 6.5: レベル 3 エディタでプログラムを保存する場合

6.2.4 レベル 4 エディタでプログラムを保存する場合

レベル 4 エディタで保存する方法を図 6.6 に示す.

Java ファイルは, エディタに書かれている内容をそのまま保存する.

Nigari ファイルは保存された Java ファイルをプログラム解析機構で解析し, C 部分に相当する内容に変更して保存する.

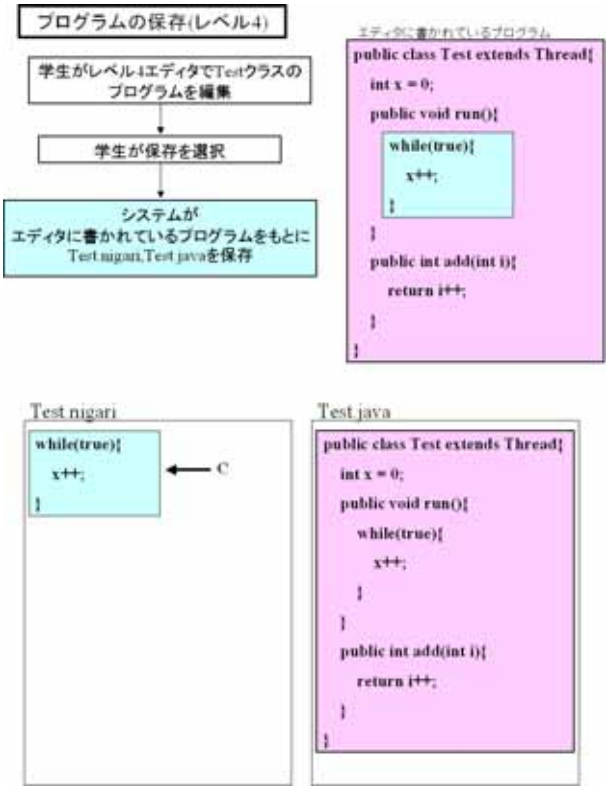

図 6.6: レベル 4 エディタでプログラムを保存する場合

6.3　プログラムを開く方法

6.3.1　レベル 1 エディタでプログラムを開く場合

レベル 1 エディタでプログラムを開く場合を図 6.7 に示す. つまり, Nigari ファイルをそのままエディタで開く.

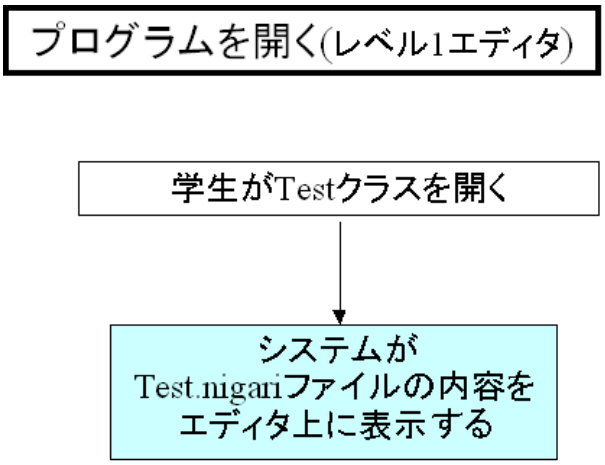

図 6.7: レベル 1 エディタでプログラムを開く場合

6.3.2 レベル 2 エディタでプログラムを開く場合

レベル 2 エディタでプログラムを開く場合を図 6.8 に示す. Java ファイルをプログラム解析機構で解析し, B 部分を変数宣言部に表示し, C 部分をメインルーチン部に表示する.

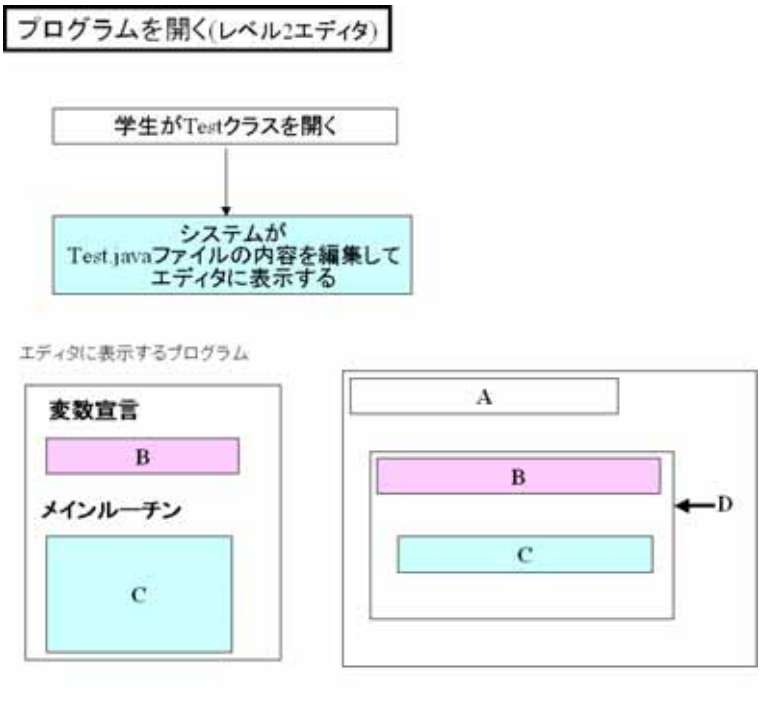

図 6.8: レベル 2 エディタでプログラムを開く場合

6.3.3 レベル 3 エディタでプログラムを開く場合

レベル 3 エディタでプログラムを開く場合を図 6.9 に示す. Java ファイルをプログラム解析機構で解析し, D 部分をエディタに表示する.

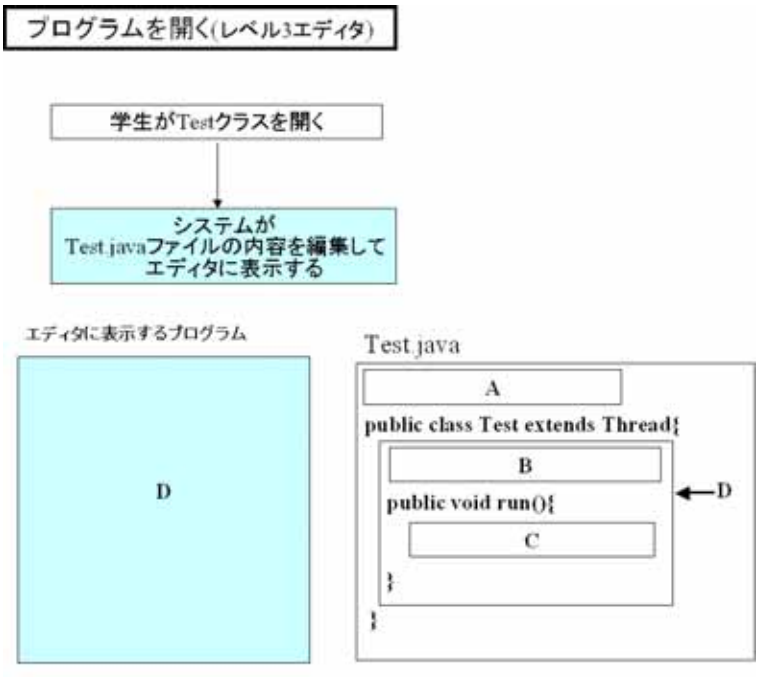

図 6.9: レベル 3 エディタでプログラムを開く場合

6.3.4　レベル 4 エディタでプログラムを開く場合

レベル 4 エディタでプログラムを開く場合を図 6.10 に示す. つまり, Java ファイルをそのまま, エディタに開く.

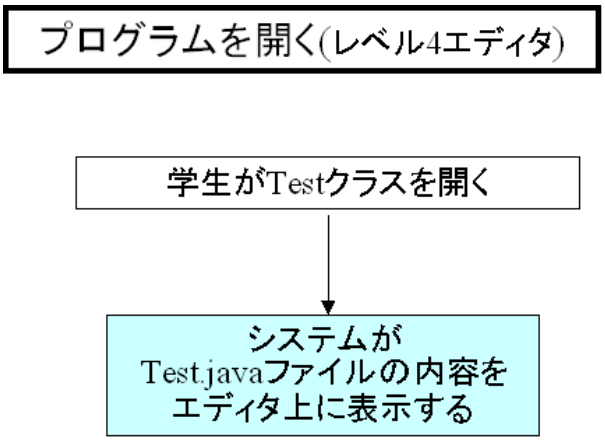

図 6.10: レベル 4 エディタでプログラムを開く場合

6.4 NJavaの可視化機能

1 レベルで書かれたユーザプログラムは Nigari 言語用の VM で動かし, 2 レベル以降で書かれたユーザプログラムは Java 言語用の VM で動かす.

Nigari 言語用の VM には, 実行位置やその時点のオブジェクトの状態情報を得る方法がある. 図 6.11 のように, NigariSystem を動かしている JavaVM は Nigari 言語用の VM から直接実行位置やオブジェクトの状態情報を調べることができる. このことから, 1 レベルのプログラムを実行したときに, 実行位置とその時点のオブジェクトの状態情報を可視化することは可能である.

2 レベル以降で Java 言語の VM を利用して実行したときに, 実行位置やその時点のオブジェクトの状態情報を得る方法を考える必要がある. NJava では, Java Virtual Machine Debug Interface[8](以後 JVMDI と略す。) を使う方法をとる.

NJava を動かしている JavaVM を NVM, ユーザプログラムを動かしている JavaVM を UVM とあらわすことにする。このとき、NVM, JVMDI, UVM の関係は, 図 6.12 のようになる。

NVM は, ユーザプログラムを JVMDI に指定する (図 6.12 の① a). すると JVMDI が UVM を作成する (図 6.12 の① b).

そして, NVM がユーザープログラムをブレークポイント (プログラムを一旦止める場所) を指定する (図 6.12 の② a). そうすると, JVMDI が UVM に, ブレークポイントまでプログラムの処理を進めるよう指示する (図 6.12 の② b).

NVM は, ブレークポイントまでユーザプログラムを実行すると, JVMDI から命令が来るまで停止している. JVMDI は, 停止している UVM に, “1 ステップ処理を進める”, “UVM のスレッド情報を要求する”, “UVM で処理されているユーザプログラムの実行位置を要求する” という三種類の命令を指示することができる. “1 ステップ処理を進める” とは, ユーザプログラムの次の 1 行を処理し停止することである.

ユーザプログラムの処理を進めたい場合は, NVM が JVMDI に 1 ステップ処理を進めるように指示を出す (図 6.12 の③ a). そして, JVMDI が UVM に命令を出し UVM が実際に 1 ステップ処理する (図 6.12 の③ b).

ユーザープログラムのオブジェクト情報がほしい場合は, NVM が JVMDI にスレッド情報を取得するよう要求を出す (図 6.12 の④ a). それに反応して, JVMDI は UVM 上にあるすべてのスレッド情報を習得する (図 6.12 の④ b). この情報を NVM で解析する. NVM にわたされたスレッドの情報は, ユーザプログラムを処理しているスレッドの情報以外も含まれている. そこで, ユーザプログラムを処理しているスレッドの情報を探し出す. 次に, 探し出したスレッドが持っている, オブジェクトの情報をすべて抜き出す. ここで, 抜き出したオブジェクトの情報にも, 本来必要としていないオブジェクトの情報も含まれている. 例えば, JavaVM を動かすために必要なオブジェクトである. これらの中から, ユーザーが知りたいオブジェクトの情報だけを抜き出す. このようにして, オブジェクトの情報を抽出することができる.

実行位置情報がほしい場合は, NVM が JVMDI に実行位置情報を要求する (図 6.12 の⑤ a). そして, JVMDI が UVM から位置情報を取得する (図 6.12 の⑤ b).

③ a-③ b, ④ a-④ b, ⑤ a-⑤ b はユーザープログラム用の JavaVM が存在し, 停止している限り何度でも繰り返すことができる. つまり, 実行位置やその時点のオブジェクトの状態情報を得ることができる. これを利用すれば, JavaVM でプログラムを実行した場合も実行位置やその時点のオブジェクトの状態情報を可視化することが可能である.

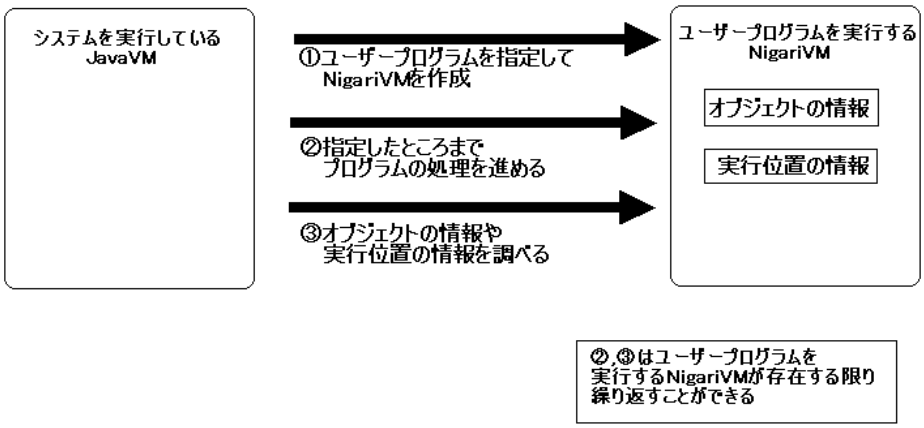

図 6.11: NigariVM から情報を得る方法

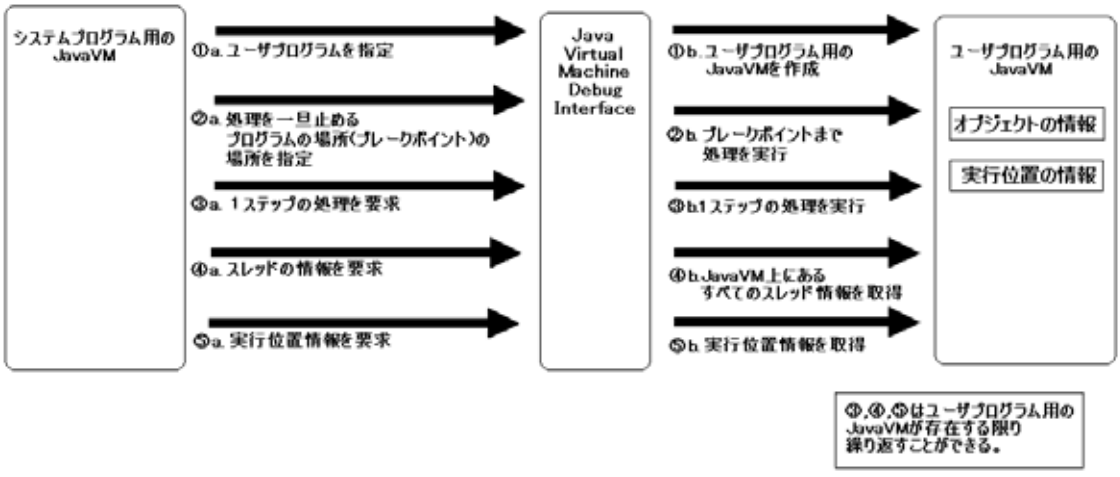

図 6.12: JavaVM から情報を得る方法

6.4.1 実行位置の可視化

実行位置の可視化の方法として, 図 6.13 のようにプログラムの処理されている行に赤い下線を引く方法をとる. Nigari では 1 行ごとに実行することが可能である. Java でも, JVMDI を用いることにより 1 行ごとに実行することが可能である.

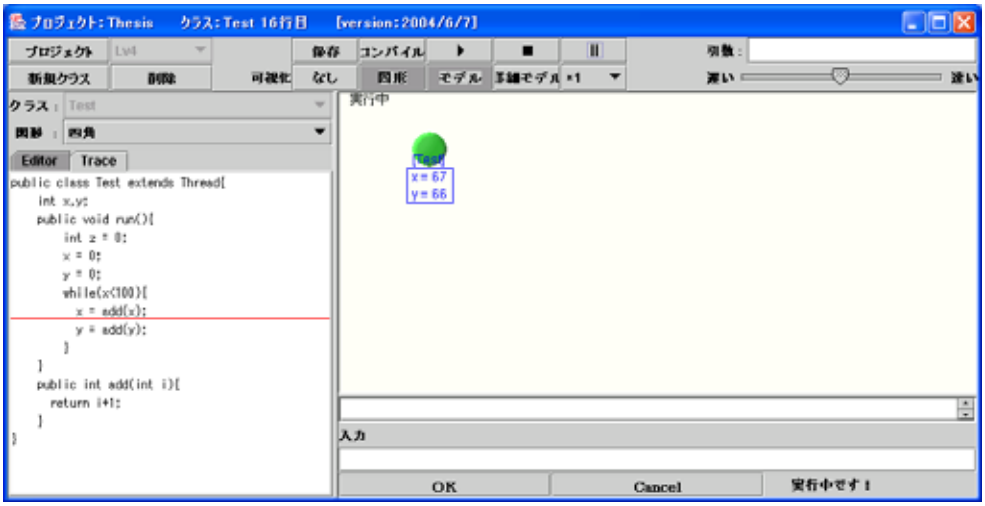

図 6.13: 実行位置の可視化

6.4.2 プログラムの振る舞いの可視化

Nigari の VM, Java の VM から抽出したオブジェクト情報は可視化用の共通のデータ構造 [1], [3] に変換する. 共通のデータ構造を持つ情報をどのように画面上に描画するかは [1], [3] を参照してほしい.

NJava でのプログラム動作可視化の種類

- **図表示**

  図 6.14 のように, NigariSystem と同じプログラムの可視化方法をとる. オブジェクトが持っている変数 x, y によって画面をオブジェクトが動き回る.

- **モデル表示**

  図 6.15 のように, オブジェクトが持っている変数, その変数に格納されている値がすべて表示される.

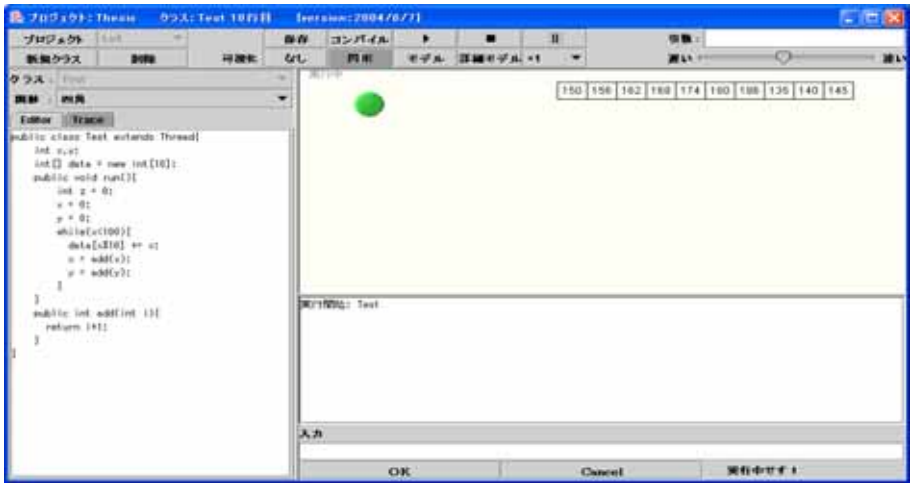

図 6.14: 図表示

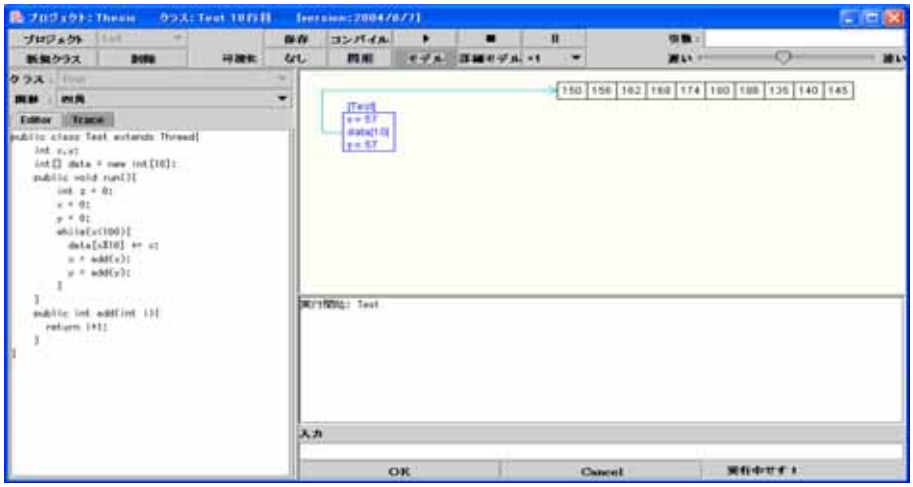

図 6.15: モデル表示

- **詳細モデル表示**

  図 6.16 のように, オブジェクトが持っている変数と, その変数に格納されている値, そのオブジェクトを動かしているスレッド, ローカル変数と格納されている値が表示される.

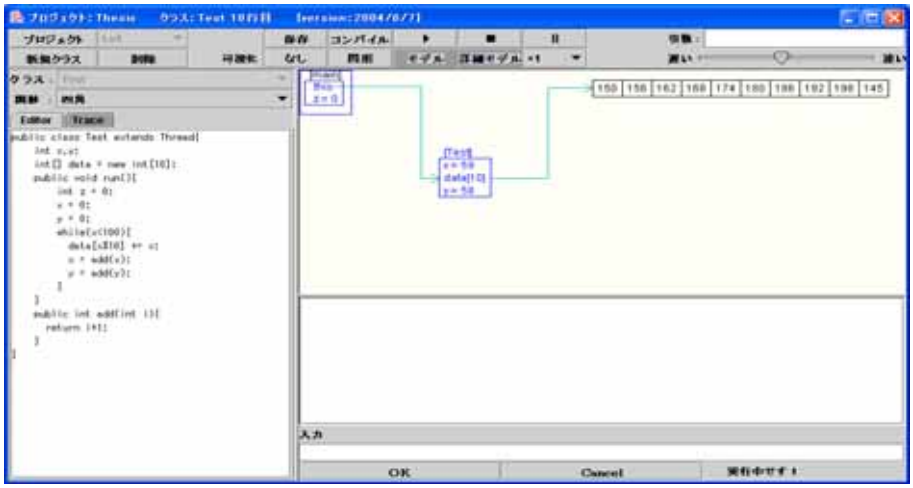

図 6.16: 詳細モデル表示

第7章　大学での試用

7.1　試用の概要

7.1.1　試用対象

- **対象学科・学年：**2004 年度早稲田大学理工学部コンピュータ・ネットワーク工学科一年 (プログラム初心者のみのクラス)
- **人数：**99 名
- **目標：**Java 言語とプログラミングの基礎の習得
- **期間・回数：**2004 年 4 月 19 日から同年 7 月 5 日までの 11 回
- **一回あたりの授業時間：** 2 時限 (約 3 時間)
- **授業形態：**各自でノートパソコンを持参. 授業 1 回ごとに説明を受け演習を行う.

7.1.2　授業構成

- **導入**
  NJava をネット上から各自の PC にダウンロードし基本的な操作方法を習得してもらう
  - 4/19　NJava の起動の仕方
- 1 **レベル**
  1 レベルで while 文や if 文などを学習してもらう. そして, 習ったことで解ける問題をいくつか出して演習してもらう.
  - 4/26　いろいろな値と while 文 (1 レベル)
  - 5/10　if 文 (1 レベル)
  - 5/17　問題集 (1 レベル)
- 2 **レベル**
  2 レベルで変数の型, 宣言, 配列オブジェクトを学習してもらう. これまでに出したレポートで正解率が悪かったものや, 勘違いが見て取れる問題を取り上げて詳しく解説をする.
  - 5/24　変数宣言と変数の型 (2 レベル)
  - 5/31　配列 (2 レベル)
  - 6/07　オブジェクト (2 レベル)
  - 6/14　レポートの解説 (2 レベル)

- **3 レベル**
  3 レベルでメソッドの定義の仕方, 使い方を学習してもらう. メソッドは理解するのに時間がかかると考え 2 コマに分けて学習する.
  - 6/21　メソッドとローカル変数 (3 レベル)
  - 6/28　メソッドの復習 (3 レベル)
- **4 レベル**
  4 レベルで Java プログラムそのものに触れてもらう. また, 直に JDK を触れてもらう.
  - 7/05　JDK とデバッグ (4 レベル)

7.1.3　評価方法

毎回の授業の最後に, 次の 4 項についてのアンケートを行った。

- 授業の分量は多かったか
- 授業の難易度はどうだったか
- 授業が楽しかったか
- 演習問題の達成度はどうだったか

また, 最後の授業には, 授業全体を通しての理解度や感想を問うアンケートを行った. 同時に次のような, システムの効果を調べるための項目もいくつか質問した.

- レベルが変わるときに書くプログラムにギャップを感じたか
- プログラムの動きが目で見えることによりデバッグが行いやすかったか

これらのアンケートから, NJava でとった方法が初心者のプログラミング学習にどのような効果があったかを調べる.

また, TA として授業に参加し, 学生の進捗状況やどのようなプログラムを書いているかをチェックする.

7.2　評価

7.2.1　NigariSystem を用いた実験の結果との比較

- **学生が感じた題材毎の難易度の比較**
  学生は前の授業の理解が不十分であると感じている場合, 次の授業中は授業を聞かずに前の授業の復習ばかりをしている傾向がある. そうして, 次第に授業の速度が速いと感じ, 遅れていく. こういった状況を生み出さないためにも, 学生が感じている難易度は上下しないほうが好ましい.

  図 7.1 をみると, NigariSystem を用いた実験では, 学生が感じている難易度は題材ごとに上下してる. しかし, NJava を用いた実験では, NigariSystem を用いた実験ほど極端な上下はなかった. このことから, 授業の速度についていけず, 遅れていってしまう学生は減っていた

と考えられる．NigariSystem を用いた授業で TA をしていたときに受けた質問の多くは，現在の授業内容の質問ではなく以前の授業内容に対する質問が多かった．しかし，NJava を用いた授業で TA をしているときに受けた質問は，大半がその時点での授業内容のものであった．

全体的に，NJava を用いて学習した人達のほうが，NigariSystem を用いて学習した人達と比べ授業全体を難しく感じている．これは，NJava を用いて学習した学生は，純粋にプログラム初心者のみを集めたクラスであるが，NigariSystem を用いて学習した学生は，プログラム初心者のみで集められたクラスではなかったためだと考えられる．

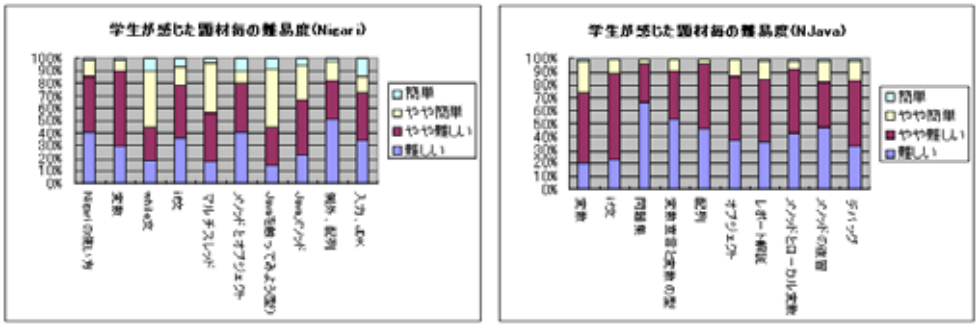

図 7.1: 難易度の比較

- **学生が感じた題材毎の楽しさの比較**

  学生がプログラムを学習していく中で，重要な要素の一つに学習意欲がある．プログラミング学習のはじめで，プログラミングは楽しくないものだという印象を植え付けることはしてはならない．学習意欲を低下させることなく，授業を進めていくことは初心者のプログラミング教育には必要である．

  図 7.2 をみると，NigariSystem を用いた授業では，Nigari 言語で学習しているうちは授業が楽しいという意見の学生の割合はほぼ一定だった．しかし，Java 言語を用いた学習が始まると右肩下がりで，授業が楽しいという学生の割合は減少している．それと比較して，NJava を用いた授業では，授業全般を通して，授業が楽しいという意見の学生の割合はほぼ一定だった．授業全般を通して学習意欲を低下させることなく，授業を進めていくことができたと考えられる．また，Java に移行していることを意識せず，スムーズに Java プログラミングに移れているように思える．ただ，学習意欲を低下させないことはできたが，向上させるにはいたっていない．学習意欲を向上させるにはどのようにすればいいかを考える必要があると考える．

- **題材毎の学生の自答率の比較**

  図 7.3 を見ると，NigariSystem を用いて学習してもらった時は，配列，メソッドを題材にした問題の自答率が飛びぬけて悪い．それと比較して，NJava を用いて学習してもらった時は，すべての題材の問題の自答率がほぼ一定である．配列，メソッドを題材にした問題の自答率が NigariSystem を用いて学習してもらった時のように，目立たなかった．このことから，配列，メソッドといった題材も他の題材と同程度の理解を得られたと考えられる．

  変数や if 文の題材で用いた問題は，NigariSystem を用いた授業と NJava を用いた授業とでほぼ同じ問題を用いた．しかし，図 7.3 を見る限り解答率に大きな差が出た．このことから，NigariSystem を用いて学習した学生の中にはプログラム初心者以外も多く含まれていたと考えられる．

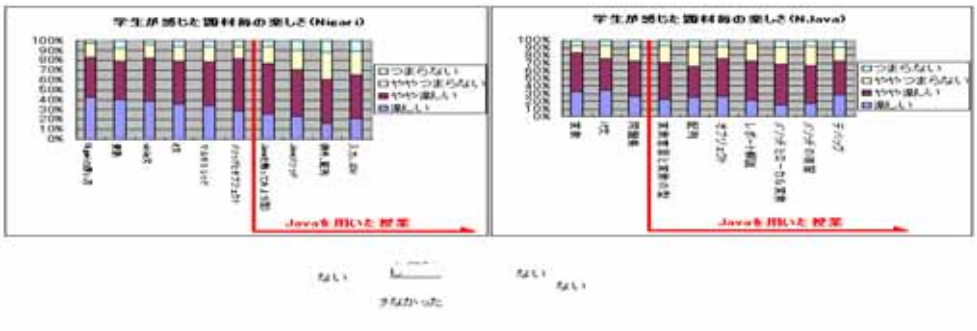

図 7.2: 楽しさの比較

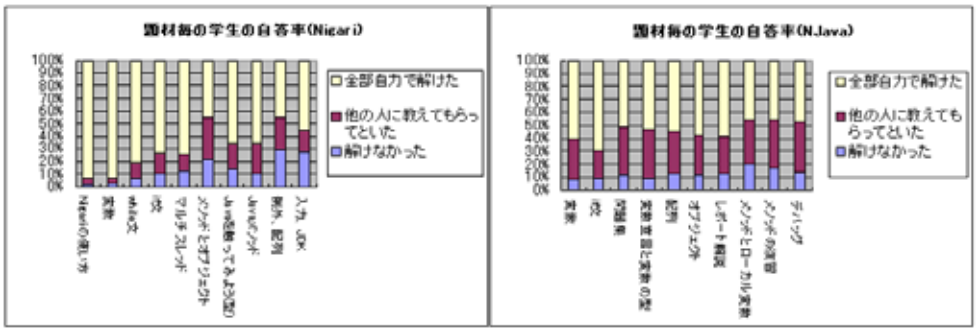

図 7.3: 自答率の比較

7.2.2 授業の TA をしながら気づいたこと

- **長いスペルの単語**
  プログラムを書き始めた初期の段階で, 学生に長いスペルの単語を書かせるのは好ましくない. 変数名やクラス名に長いスペルの単語を使う問題を出題すると, 多くの人がスペルミスにつまずいてしまう. 多くの学生が, コンパイラのエラーの説明を読んでも意味がわからず, 何が間違っていてコンパイラがエラーを出しているかがわからない状況に陥っていた。

- **コンパイルエラーへの対応**
  プログラムを学習し始めたばかりの頃は, コンパイルエラーが出る原因がわからないため, 学生が TA に質問することは仕方がない. しかし, コンパイルエラーがでたら TA に質問をするという姿勢が染み付いてしまう学生がいる. 学習が進んでいっても, コンパイルエラーが出るとすぐに TA に質問をして, 自分で修正をしようとしない.

- **問題のヒントの趣旨をつかめない**
  問題を出題するとき, 問題が難しいと判断した場合, ヒントを添えるようにしていた. しかし, 特定の問題に対し, なぜそのヒントを出したかという意図をつかめない人が数多くいた.

- **質問を多くする学生と, その学生の習得具合の関係**
  質問を多くする学生だからといって, 自分で積極的に学習を進めているから多くの疑問が生

まれ質問してくるのだと思うのは教えている側の勝手な思い込みである．こういった学生の多くは, 物事を考える習慣がなく, 自分が知らなかったり, わからなかったりすることはすぐ人に聞くという習慣が身についてしまっている．そのせいか, 応用問題にはまったく手がつけられない様子だった．

7.2.3 プログラムの動きを目で追えるようにしたことに対する評価

プログラムの振る舞いの可視化

プログラムの振る舞いを可視化した効果についての詳細は, [1], [3] を参照してほしい．

学生は, プログラムを学習し始めたばかりのころは, 図 6.14 のような図表示でプログラムの振る舞いを確認していた．しかし, プログラムの学習が進むにつれて, 図表示でプログラムの振る舞いを確認する学生が減っていった．そして, オブジェクトのすべての情報を確認することができる, 図 6.15 のようなモデル表示でプログラムの振る舞いを確認する学生が増えていった．やはり, プログラムが複雑になってくるとプログラムを実行している間のすべての情報が, 目で確認できるほうがプログラムの振る舞いを確認しやすかったように思える．

実行位置の可視化

実行位置を可視化した理由は, プログラムが実行されているときに, プログラムのどの部分が実行されてオブジェクトにどのような影響を及ぼしているのかを学生が確認できるようにするためである．この機能の効果はを確認するには, 学生にプログラムのデバッグをしてもらってこの機能を使わせた上で意見を聞くことがよいと考えた．

図 7.4 が NJava がデバッグをする際に役立ったかを学生に聞いた結果である．約 80 %の学生が NJava の可視化機能を使うとデバッグすることが容易だったといってくれている．このことから, 実行位置を可視化することはプログラムのどの部分が実行されてオブジェクトにどのような影響を及ぼすかを確認できるようにするのに効果的であるといえる．

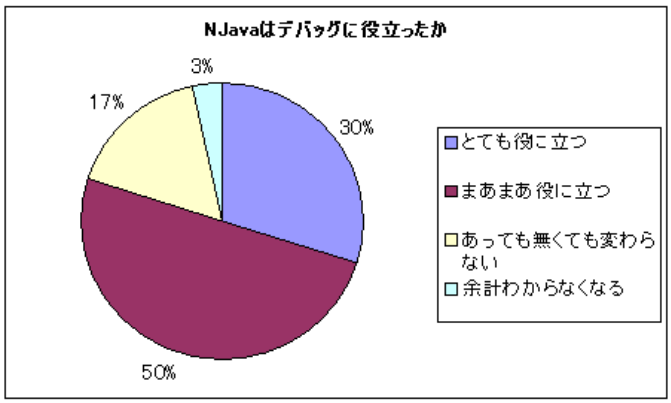

図 7.4: NJava デバッグに役立ったか

第8章　4つのレベルに分けたプログラミング学習の評価

8.1　1レベルから2レベルへ移行したときのギャップ

NJavaの1レベルと2レベルの違いは, 変数宣言部を書かなければいけないことである. このことから, NJavaを用いた学生が1レベルから2レベルに移行した時に感じたギャップと, NigariSystemを用いた学生がJava言語で変数宣言を習ったときに感じたギャップとを比較した. NigariSystemを用いた学生のJava言語へのギャップとNJavaを用いた学生のJava言語へのギャップの比較 (変数宣言) が図8.1である.

図8.1を見る限り, NigariSystemを用いた実験に参加した学生と, NJavaを用いた実験に参加した学生が変数宣言に感じたギャップに差はない. ただ, NJavaを用いた実験では, 変数宣言と一緒に配列についても学習してもらっているため, 難易度は上がっていたはずである. それにもかかわらず, NigariSystemを用いた実験に参加した学生と, NJavaを用いた実験に参加した学生が変数宣言に感じたギャップに差がなかった. これは配列を学習してもらったタイミングが適切であったためだといえる.

2レベルという段階をおいた最も大きな理由は配列である. この段階を追った学習方式が, 配列を理解するのに効果があったかを調べる. 図8.2がNigariSystemを用いて学習した学生と, NJavaを用いて学習した学生の配列に関する問題の自答率の比較である. NigariSystemを用いた実験に参加した学生の約45 %が自力で問題を解いているのに対して, NJavaを用いた実験に参加した学生の約55 %が自力で問題を解いている. また, NigariSystemを用いた実験に参加した学生の約30 %が問題が解けなかったと答えているのに対し, NJavaを用いた実験に参加した学生の約10 %が問題が解けなかったと答えている. これらから考えて, 配列の理解度は上がっていると考えられる. さらに, 配列がまったくわからないという学生も減少しているといえる.

8.2　2レベルから3レベルへ移行したときのギャップ

2レベルから3レベルに移行したときに増えることは, “public”, “static” などのような, メソッドやクラスの宣言である. 変数でも “public”, “static” を用いることができるが, 今回の実験では変数を学習する時点では教えていない. そのため, メソッドの定義を行うことができる3レベルでこれらがはじめて出てくることになる. このことから, NJavaを用いた学生が2レベルから3レベルに移行した時に感じたギャップと, NigariSystemを用いた学生がJava言語でメソッドを習ったときに感じたギャップとを比較した. NigariSystemを用いた学生のJava言語へのギャップとNJavaを用いた学生のJava言語へのギャップの比較 (Java特有の記述) が図8.3である.

図8.3を見る限り, NigariSystemを用いた実験に参加した学生の80 %以上がJava特有の記述にギャップを感じている. NJavaを用いた実験に参加した学生の約50 %がJava特有の記述にギャップを感じている. メソッドの記述方法に関しては少しではあるがギャップを緩和できているといえる.

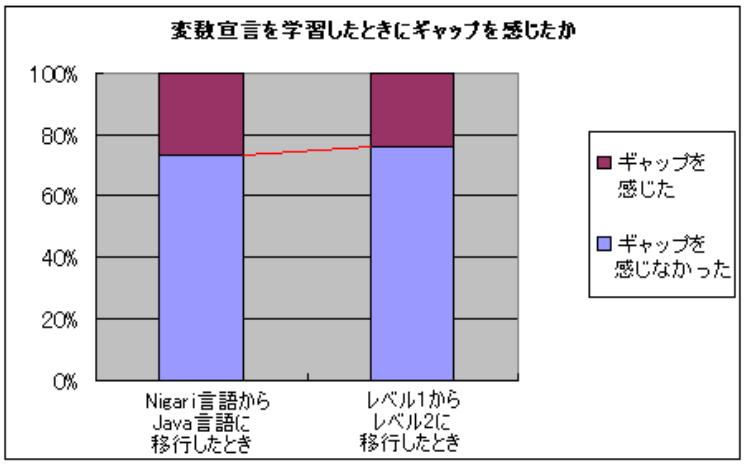

図 8.1: NigariSystem を用いた学生の Java 言語へのギャップと NJava を用いた学生の Java 言語へのギャップの比較 (変数宣言)

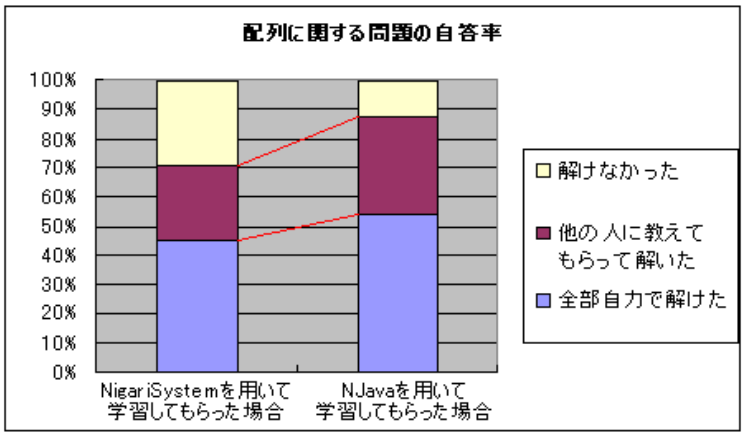

図 8.2: 配列に関する問題の自答率

3 レベルという段階をおいた最も大きな理由はメソッドである．この段階を追った学習がメソッドを理解するのに効果があったかどうかを調べる．図 8.4 が NigariSystem を用いて学習した学生と，NJava を用いて学習した学生の配列に関する問題の自答率である．自分で解けた人の割合，他人に聞いて解いた人の割合，解けなかった人の割合がほとんど一緒である．このことから，メソッドを理解することにはあまり効果がなかったようである．

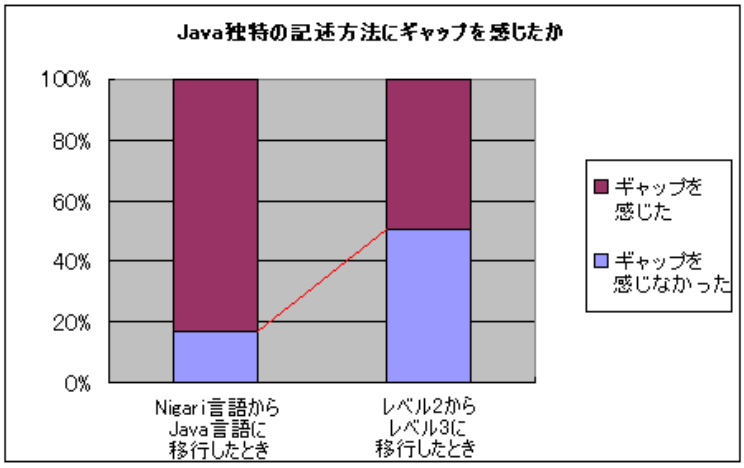

図 8.3: NigariSystem を用いた学生の Java 言語へのギャップと NJava を用いた学生の Java 言語へのギャップの比較 (Java 特有の記述)

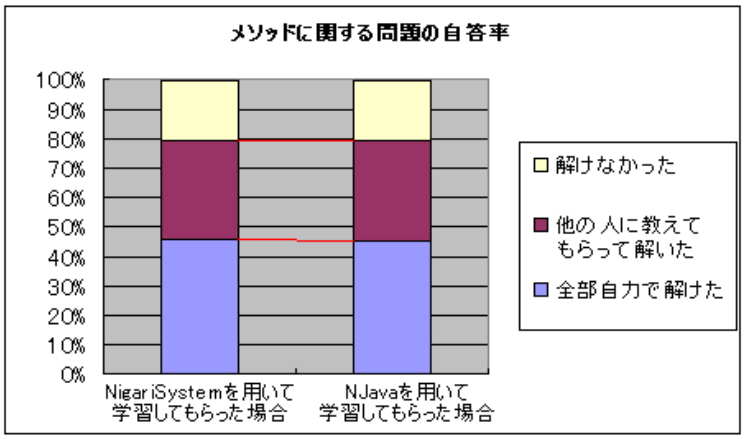

図 8.4: メソッドに関する問題の自答率

第9章　考察と改良すべき点

9.1　授業スタイル

すべての授業が終わった後, 学生に授業のスタイルとしていいと思うものを答えてもらった. 図 9.1 がその結果である. ほぼすべての学生が NJava を用いた学習が授業スタイルとして好ましいと答えている. しかし, 半数以上の学生が NJava を用いるだけでなく, JDK を授業の中に加えてほしいと答えている. このことから, NJava を使って学習しながら JDK の使い方を学ぶ仕組みがあれば, より満足感のある授業になると考える.

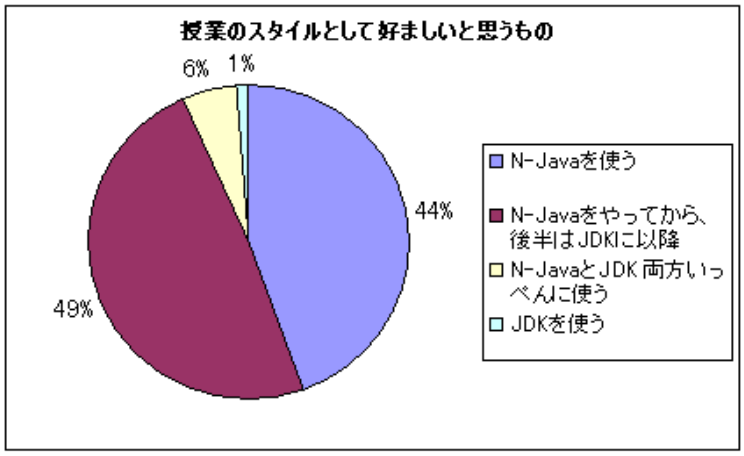

図 9.1: 授業のスタイルとして好ましいと思うもの

9.2　Java 言語の理解

図 9.2 は NigariSystem を使って学習した学生と, NJava を使って学習した学生がそれぞれのシステムを使うことにより, Java の理解が深まったと思うかをたずねた結果である. NigariSystem を用いて学習した学生のうち, 40 %弱の学生が NigariSystem を用いることにより Java の理解が深まったと答えているのに対し, NJava を用いて学習した学生のうち, 約 80 %の学生が NJava を用いることにより Java の理解が深まったと答えている. NigariSystem を用いて学習した学生で, “かえってわからなくなった”, “あってもなくてもかわらない” と答えた人たちは, Nigari 言語と Java 言語はまったく違うものであると感じていた. そのため, NJava では Nigari 言語, Java 言語というふうに, 二つの言語を学ぶということを意識させないようにした. この効果があり, NJava を用いて学習した学生の多くが Java の理解が深まったと答えたと思える.

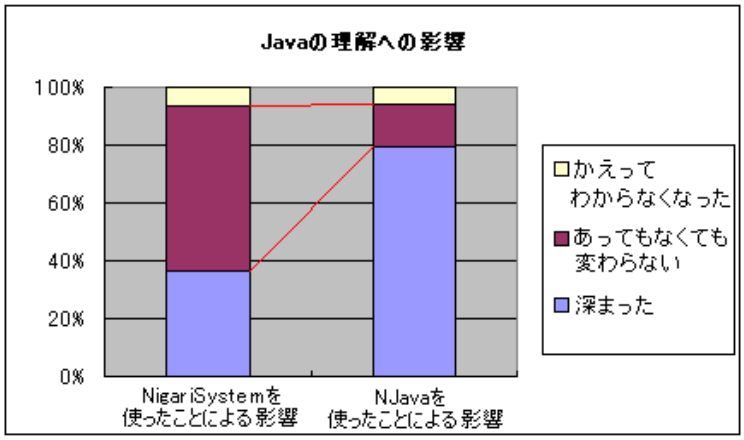

図 9.2: Java 言語の理解への影響

謝辞

本研究では, 全般にわたり筧捷彦教授の指導のもとで行われました. 時には叱咤激励し, 時にはあたたかくみまもってくださいました筧教授の熱心なご指導に, この場を借りまして心より感謝いたします.

NJava を用いた授業で, 学生に丁寧でわかりやすい解説, 説明を行ってくださった甲斐宗徳先生にも, 深く感謝の意を表します.

また同様に, 私の相談にもいやな顔一つせず気軽にのっていただき, たくさんの参考意見を下さった長慎也先輩にもお礼申し上げます.

最後になりましたが, 一緒に研究をした前島真一君, 小田嶋祐介君, 授業の TA を手伝ってくれた佐々木康太朗君, NJava を用いた授業に参加してくれた一年生たちも, ありがとうございました.

私自身, 真剣に取り組んでまいりましたが, この研究の成果は本当に皆様の援助があってこその賜物だと思います.

参考文献

[1] 小田嶋祐介：プログラミング入門システム Nigari の可視化機能の改良
2003 年度早稲田大学理工学部情報学科卒業論文

[2] 川合晶：プログラミング言語教育用 Nigari 言語と Nigari System の構築と実装
2002 年度早稲田大学理工学部情報学科卒業論文

[3] 鈴木健司：プログラミング入門システム NigariSystem の可視化機能拡張
2003 年度早稲田大学理工学部情報学科卒業論文

[4] 長慎也, 甲斐宗徳, 川合晶, 前島真一, 筧捷彦：Nigari ― Java 言語へも移行しやすい初学者向けプログラミング言語
情報処理学会論文誌　研究報告 「コンピュータと教育」 No.071 - 003

[5] 長慎也, 甲斐宗徳, 川合晶, 前島真一, 筧捷彦：プログラミング環境 Nigari ―初学者が Java を習うまでの案内役
情報処理学会論文誌 Vol.45No.SIG9(PRO22)

[6] 日野孝昭：プログラミング入門システム Nigari の設計と実装
2002 年度早稲田大学理工学部情報学科卒業論文

[7] 前島真一：プログラミング教育用システム Nigari の構築と実装
2002 年度早稲田大学理工学部情報学科卒業論文

[8] JavaTM Virtual Machine Debug Interface Reference
http://java.sun.com/j2se/1.5.0/docs/guide/jpda/jvmdi-spec.html